The SVT-AV1 encoder: overview, features and speed-quality tradeoffs

Faouzi Kossentini, Hassen Guermazi, Nader Mahdi, Chekib Nouira, Amir Naghdinezhad, et al.

Faouzi Kossentini, Hassen Guermazi, Nader Mahdi, Chekib Nouira, Amir Naghdinezhad, Hassene Tmar, Omar Khlif, Phoenix Worth, Foued Ben Amara, "The SVT-AV1 encoder: overview, features and speed-quality tradeoffs," Proc. SPIE 11510, Applications of Digital Image Processing XLIII, 1151021 (21 August 2020); doi: 10.1117/12.2569270

SPIE.

Event: SPIE Optical Engineering + Applications, 2020, Online Only

The SVT-AV1 Encoder: Overview, Features and Speed-Quality Tradeoffs

Faouzi Kossentini, Hassen Guermazi, Nader Mahdi, Chekib Nouira, Amir Naghdinezhad, Hassene Tmar, Omar Khlif, Phoenix Worth, Foued Ben Amara

Intel Corporation, Suite 700, 450 SW Marine Drive, Vancouver, BC V5X 0C3 Canada

ABSTRACT

The Scalable Video Technology AV1 (SVT-AV1) encoder is an open-source software AV1 encoder that is architected to yield excellent quality-speed-latency tradeoffs on CPU platforms for a wide range of video coding applications. The SVT-AV1 encoder is based on the SVT architecture, which supports multi-dimensional parallelism, multi-pass partitioning decision, multi-stage/multi-class mode decision, and multi-level spatiotemporal prediction and residual coding algorithms. Given a latency constraint, the SVT-AV1 encoder maximizes the CPU utilization on multicore CPUs, through picture-based parallelism for high-latency video applications, and through segment-based parallelism for low-latency video applications. The picture-level/segment-level parallelism allows the SVT-AV1 encoder to produce identical bit streams, irrespective of whether single- or multi-threaded encoding is performed. In mode decision, the SVT-AV1 encoder yields many speed-quality tradeoffs with high granularity, mainly through multi-pass processing of each superblock and multi-stage/multi-class processing of each block, and through the different levels of the prediction/coding features. The resulting speed-quality tradeoffs of SVT-AV1 are compared, in Video-On-Demand (VOD) use-cases, to those of libaom (another open-source AV1 encoder), and to those of the latest x264 (AVC), x265 (HEVC) and libvpx (VP9) open source encoders.

Keywords: Video coding, AV1, multithreading, segments, multi-pass partition decision, multi-stage mode decision.

1. INTRODUCTION

Over the past decade, a significant number of applications have emerged that require the sharing and/or consumption of visual content and experiences, leading to the accelerated growth in video data and corresponding video traffic. Examples of such applications are over-the-top linear video streaming, live broadcast of user-generated video content, media analytics and cloud gaming. Many of the recently developed visual compression technologies and standards (e.g. AVC [1], HEVC [2], VP9 [3] and AV1 [4,5]) achieve high compression efficiency, however, standard-compliant encoders can be very complex, requiring large computational and memory resources. Thus the objective is to achieve the best-possible complexity-quality tradeoffs, subject to the application-specific constraints (e.g., latency). Towards such an objective, a software-based video encoder is expected to navigate a landscape of many conflicting requirements and provide gradual transitions in complexity-quality tradeoffs. The Scalable Video Technology AV1 (SVT-AV1) encoder is an open-source software AV1 encoder that was developed to include architectural capabilities and algorithmic features that would enable it to address efficiently and effectively the various requirements of the different video processing/coding applications.

The SVT-AV1 encoder supports multi-dimensional parallelism, multi-pass partitioning decision, multi-stage and multi-class mode decision, and multi-level spatiotemporal prediction and residual coding algorithms. With respect to the multi-dimensional parallelism support, the multi-core CPU utilization of the SVT-AV1 encoder can be maximized, given a latency constraint, via picture- and segment-based parallelism for high-latency applications, and only segment-based parallelism for low-latency applications. These two types of parallelism allow the SVT-AV1 encoder to produce the same bit stream regardless of whether single- or multi-threaded encoding is employed. Moreover, the superblock-based multi-pass partitioning allows the SVT-AV1 encoder to search the best partition for each superblock in multiple passes, where each subsequent pass brings the encoder closer to the best possible partition. For each partitioning pass, all blocks of each considered partition are processed through multiple stages, with decreasing number of candidates and increasing accuracy in each subsequent stage. Each stage involves multiple classes, with each class corresponding to a prediction type. Moreover, for each stage/class, SVT-AV1 supports multiple complexity levels for any given prediction or residual coding feature, with each level designed to exploit the input video's spatiotemporal characteristics and the SVT and AV1 structural

Applications of Digital Image Processing XLIII, edited by Andrew G. Tescher,
Touradj Ebrahimi, Proc. of SPIE Vol. 11510, 1151021 · © 2020 SPIE
CCC code: 0277-786X/20/$21 · doi: 10.1117/12.2569270

constraints. Such a highly flexible architecture permits the SVT-AV1 encoder to implement any AV1 prediction or residual coding feature almost anywhere in the encoder, and with virtually any accuracy level for any given pass-stage-class combination.

As a result of the unique architecture- and algorithms-based SVT features, the SVT-AV1 encoder can achieve not only very-high processing granularity, but also the best-possible complexity-quality tradeoffs, given the application constraints.

The rest of the paper introduces the main architectural and algorithmic features/characteristics of the SVT-AV1 encoder and presents some comparative simulation results. Section 2 presents the main characteristics of the SVT architecture and the multi-dimensional parallelism of the encoder. The multi-pass partitioning and multi-stage/multi-class processing in mode decision are then discussed in Sections 3 and 4, respectively. Section 5 presents the major computationally efficient SVT algorithms. Section 6 presents the speed-quality tradeoffs of the encoder for 10-bit content, relative to those of the libaom AV1 encoder, and for 8-bit content, relative to the libaom, x264 (AVC), x265 (HEVC) and libvpx (VP9) encoders. Some conclusions are presented in Section 7.

2. SVT-AV1 ENCODER ARCHITECTURE

The SVT architecture is designed to maximize the utilization of the available computational resources to achieve the best-possible speed-quality-latency tradeoffs. It is based on three key ideas: (1) Splitting the encoding operation into a set of independent encoding processes, where partition/mode decisions and normative encoding/decoding are decoupled, and where the degree of parallelism in each process can be controlled through the use of segments-based parallelism and/or multi-precision-level features; (2) Adopting a prediction structure that allows for efficient picture-level parallelism; and (3) Splitting each picture into segments, and then processing multiple segments of the picture in parallel (while satisfying inter-dependency constraints) to achieve better utilization of the computational resources with no loss in video quality. These design aspects of the SVT architecture are at the core of the SVT-AV1 encoder's ability to scale appropriately with, and fully exploit, any additional computing resources (e.g., more cores, higher frequency, bigger cache/bandwidth). An overview of the encoder design is presented next, followed by a discussion of the picture-level and segment-level parallelism algorithms in the encoder.

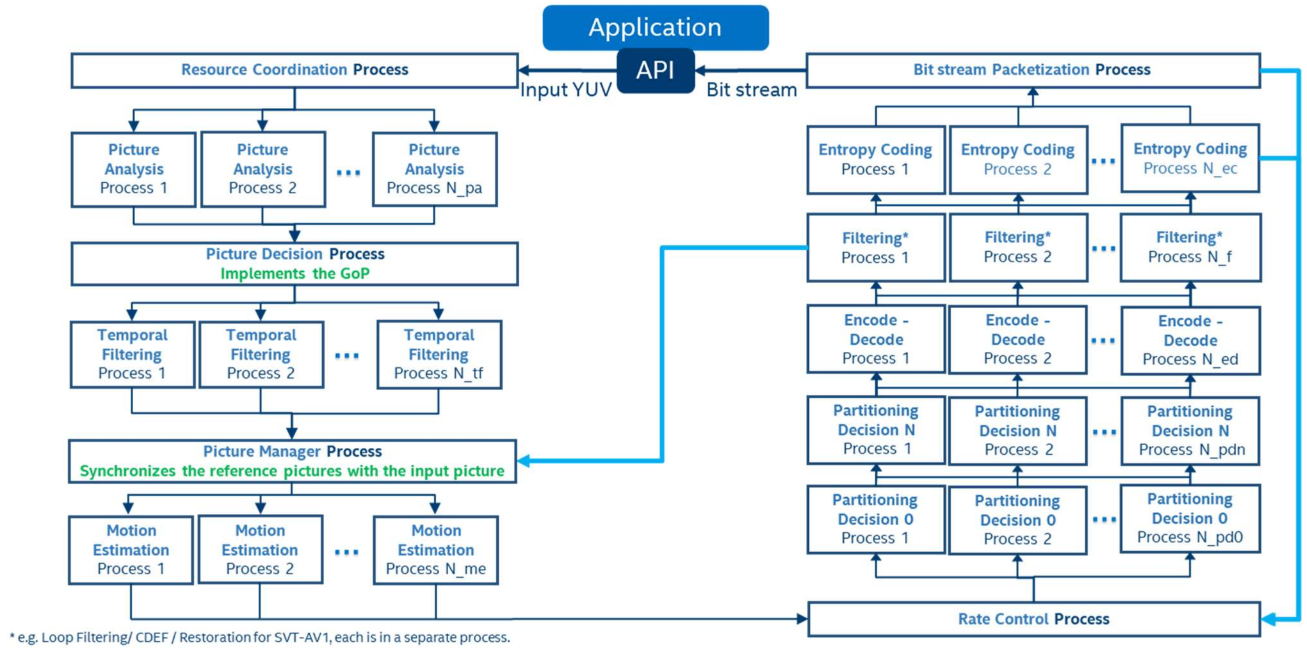
2.1 Encoder processes

As shown in Figure 1, the general encoder flow is divided into processes. A process is an execution thread in software or an IP core in hardware. Processes execute one or more encoder tasks (e.g., motion estimation, rate control, deblocking filter, etc.). The processes are either picture-based control-oriented processes or data-oriented processes. Two examples of control-oriented processes are the Picture Decision Process, which determines the prediction structure, and the Picture Manager Process, which determines when to start processing/encoding a picture depending on the state of the decoded reference picture buffer. An example of a data-oriented process is the Motion Estimation (ME) Process, which performs motion estimation using the input or the reconstructed picture data. Only one process of each control process is allowed, since control decisions cannot be made independently of each other without risking data leakage and/or deadlock scenarios.

To facilitate parallel processing on a variety of computing platforms (e.g., multi-core CPUs, GPUs, DSPs, FPGAs), inter-process communication should be minimized. One important design feature of the SVT-AV1 encoder architecture is that processes are stateless. All information related to process states is conveyed via inter-process control and data information. The system resource managers manage inter-process control and data information coherence through FIFO buffers.

The architecture is flexible enough to support an implementation in which one superblock (SB) at a time is encoded through the pipeline starting at the ME process up to and including the Encode/Decode (Enc/Dec) process. Alternatively, a single encoder task might process all SBs in each picture before proceeding to the next task. In the latter case, for example, ME could be performed on all SBs within a picture before moving to the next task. Such motion information, as well as picture statistics that are important for rate control, are gathered early in the processing pipeline and used in subsequent tasks. To address/support low-delay video applications, the rate control process may proceed before all ME picture processes are completed. The picture quantization parameter value would then be derived using all available information at the time of the derivation. The next major processing step involves the processes associated with the coding loop. The latter includes tasks such as intra prediction, mode decision, transform and quantization. Subsequent processing would involve in-loop deblocking filtering, constrained directional enhancement filter (CDEF) and restoration filtering, followed by entropy coding and packetization.

An important aspect of the SVT encoder architecture is that it supports multiple encoder instances. That is, the SVT-AV1 encoder that is implemented based on the architecture described in this paper is capable of encoding a 720p video sequence and another 1080p video sequence, simultaneously. At any given time, some of the ME, coding loop, and other processes would be handling the 720p video sequence while other processes would be handling the 1080p video sequence.



Application
API
Input YUV
Bit stream
Resource Coordination Process
Picture Analysis Process 1
Picture Analysis Process 2
Picture Analysis Process N_pa
Picture Decision Process
Implements the GoP
Temporal Filtering Process 1
Temporal Filtering Process 2
Temporal Filtering Process N_tf
Picture Manager Process
Synchronizes the reference pictures with the input picture
Motion Estimation Process 1
Motion Estimation Process 2
Motion Estimation Process N_me
Bit stream Packetization Process
Entropy Coding Process 1
Entropy Coding Process 2
Entropy Coding Process N_ec
Filtering* Process 1
Filtering* Process 2
Filtering* Process N_f
Encode - Decode Process 1
Encode - Decode Process 2
Encode - Decode Process N_ed
Partitioning Decision N Process 1
Partitioning Decision N Process 2
Partitioning Decision N Process N_pdn
Partitioning Decision 0 Process 1
Partitioning Decision 0 Process 2
Partitioning Decision 0 Process N_pd0
Rate Control Process
* e.g. Loop Filtering/ CDEF / Restoration for SVT-AV1, each is in a separate process.

Figure 1. High-level SVT-AV1 encoder process dataflow.

2.2 Picture-level parallelism

The SVT-AV1 encoder allows for the video pictures to be organized into periodic groupings, where each group is referred to as a Group of Pictures (GoP). The GoP itself consists of several periodic mini-GoPs. The pictures in a mini-GoP are organized into a hierarchical prediction structure, where each reference picture serves as reference for another subset of pictures. The latter in turn serve as reference pictures for another subset of pictures, etc. As a result, the encoding of pictures in each subset could proceed simultaneously as soon as the reference pictures become available, hence the picture-level parallelism feature of the SVT-AV1 encoder. Figure 2 shows an example of the relationships between pictures contained in a five-layer prediction structure, where each picture references at most one picture in each direction. In the example shown in Figure 2, pictures 0 and 16 are said to belong to temporal layer 0 or base layer, whereas pictures 1, 3, 5, 7, 9, 11, 13 and 15 are said to belong to the non-reference layer, or temporal layer 4. In this case, the parallel processing of the pictures could proceed as follows: 0 → 16 → 8 → (4||12) → (2 || 6 || 10 || 14) → (1 || 3 || 5 || 7 || 9 || 11 || 13 || 15), where || indicates that pictures could be processed in parallel. This example of picture-level parallel processing could be achieved, for example, by simultaneously applying multiple instances of the same process to different pictures. Inherent in the picture-level parallelism approach is a relatively high latency due to the need to process many pictures before the first picture following the Intra picture is encoded and output in the bit stream. Moreover, picture-level parallelism comes at the expense of a significant increase in memory, which would be needed to store the reference (reconstructed) pictures.

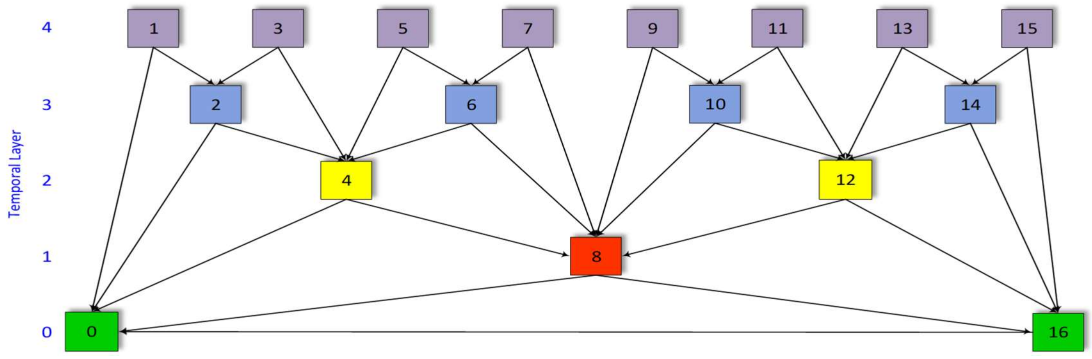
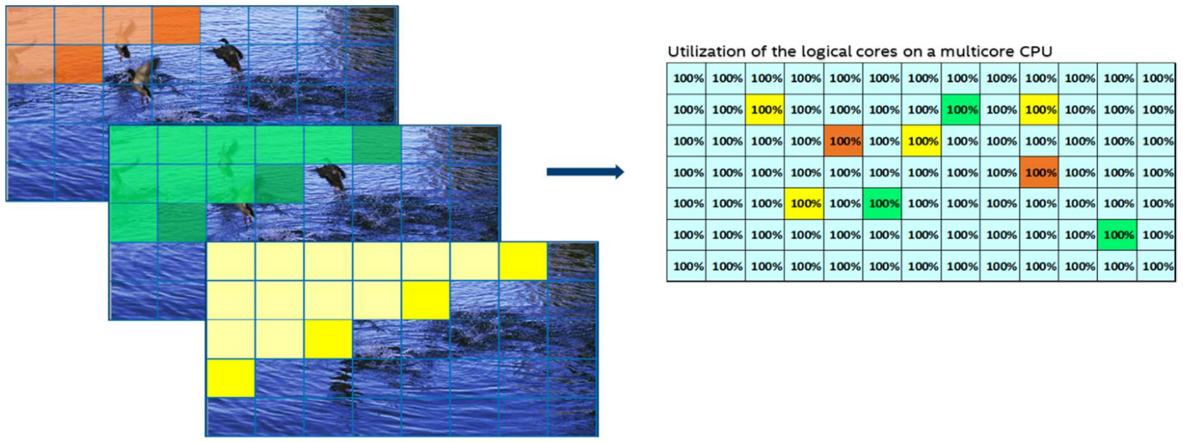


Figure 2: Five-layer prediction structure with at most one reference picture in each direction.

2.3 Segment-level parallelism

Picture-level parallelism may not be enough to fully utilize the available resources, as the encoding of a given picture must wait for the reference pictures associated with that picture to be encoded. A better utilization of the computational resources could be achieved if the processing of a given picture would be performed by processing different parts of the picture simultaneously. One sub-picture-based parallelism method is described next. First, the picture is split into smaller parts, or segments. A segment is a rectangular-shaped area of the picture consisting of several contiguous SBs. An example of such splitting is shown in Figure 3, where each of the pictures is split into 40 segments. All such segments could then ideally be processed by different processing cores simultaneously. However, there are several dependencies between the segments in the picture, in that the processing of a given SB in each segment would require that the left, top left, top and top right neighboring SBs be processed first. As a result, the processing of the segments would then need to be performed in a wavefront manner. In the extreme case where each segment represents an SB, the processing of a given segment would not start until the left, top left, top and top right neighboring segments are processed. In Figure 3, the lightly-shaded segments are the segments that have already been processed, and the remaining dark-shaded segments would be the segments that could be processed in parallel. It follows then that, in this arrangement, segments could be processed in parallel without any loss in video quality, i.e., the resulting bit streams would be the same regardless of the number and size of the segments of the employed configuration, for as long as the encoding dependencies are respected. Although segment-based parallelism helps in reducing the latency in the encoder through a more efficient utilization of the computational resources, it may still result in a large memory footprint for the encoder.



100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Figure 3. Example of segment-level parallelism. Each segment is processed using a core on a multi-core processor. The dark orange, green and yellow cores are the active cores processing the corresponding segments in the pictures on the left. The cyan blocks correspond to cores processing other segments from other pictures.

3. SUPERBLOCK-BASED MULTI-PASS PARTITIONING DECISION

Figure 4 shows an example of a possible SB partition, where each parent square block could be further subdivided into square/rectangular blocks following one of the eight non-square partitions shown in Figure 5. Given the very-large number of possible SB partitions, it would be computationally very expensive to evaluate the tradeoffs of all such possible SB partitions. Consequently, a multi-pass Partitioning Decision (PD) algorithm is employed in SVT-AV1, as illustrated in Figures 6-7. The PD algorithm performs multiple PD passes, starting with PD_Pass_0, and all the way to PD_Pass_N. The input to PD_Pass_0 consists of all allowed SB partitions, and the output consists of the best-possible SB partition (such as the one in Figure 4), given the tested non-square partitions (e.g., one of the partitions in Figure 5), modes (e.g., intra, inter) and features (e.g., Tx type search, Interpolation filter search). The next PD pass (PD_Pass_1) performs a PD refinement. More specifically, it tests for the best SB partition, the parent square block, the square blocks that are one size larger than that of the current square block, and the square blocks that are one size smaller than that of the parent square block. The output of PD_Pass_1 is also the best-possible PD partition given the tested non-square partitions, modes and features. The algorithm is designed to test more non-square partitions, more modes, and more features in PD_Pass_1, relative to PD_Pass_0. Each subsequent PD pass performs the same refinement, but with higher decision accuracy, through the testing of more non-square partitions, modes and features. The decision accuracy increases with each additional PD pass, until the last PD_Pass_N, which is designed to test the most modes and features.

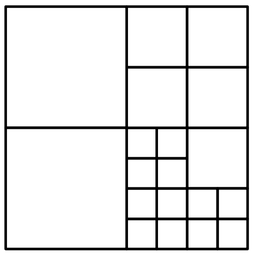


Figure 4. Example of a superblock partition.

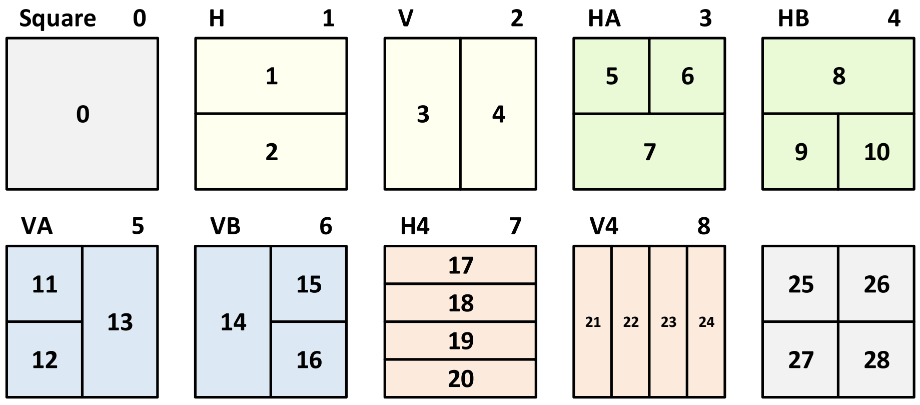


Figure 5. Possible partitions for a square block in AV1. Some restrictions on the allowed partitions apply depending on the parent square block size.

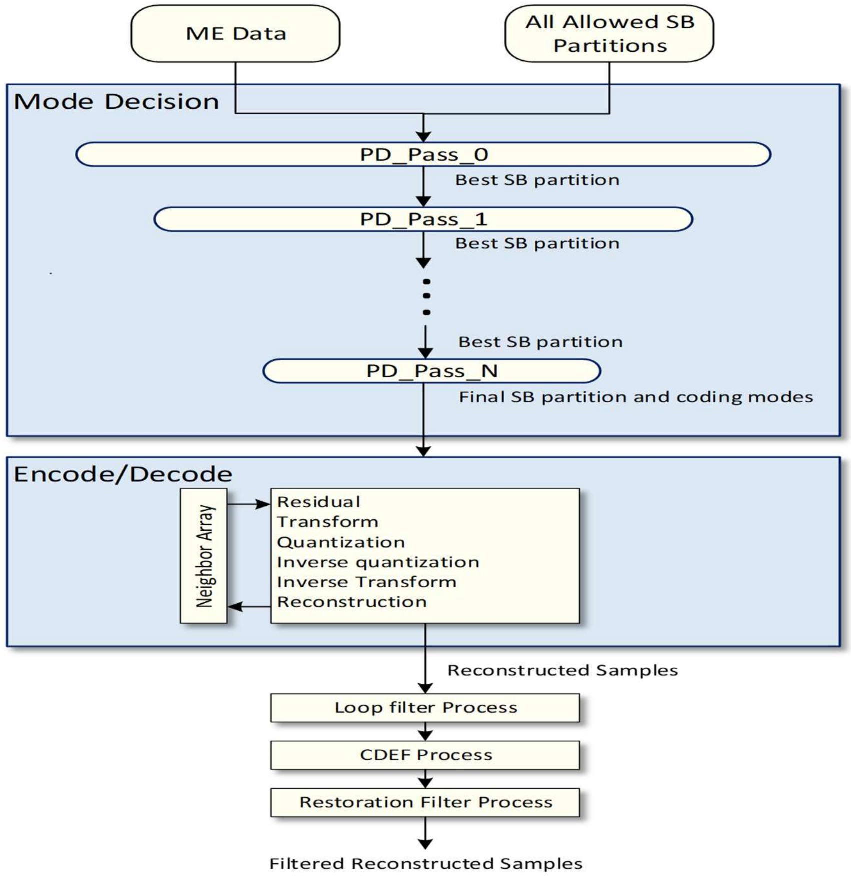


Figure 6. Multi-pass Partitioning Decision.

An illustration of the different processing details that can take place in each PD pass is given in Figure 7. In this example, PD_Pass_0 is based on the ME and intra DC prediction data. The prediction candidate cost is used to select the best SB partition output from PD_PASS_0. PD_PASS_1 may involve more complex partitions, higher-precision prediction tools, and more accurate features, in order to produce a more-accurate SB partition. For example, PD_Pass_1 may use more accurate interpolation filters in the sub-pel search. In PD_Pass_N, the fullest set of partitions, prediction features, and coding tools are used to determine the final SB and associated coding modes.

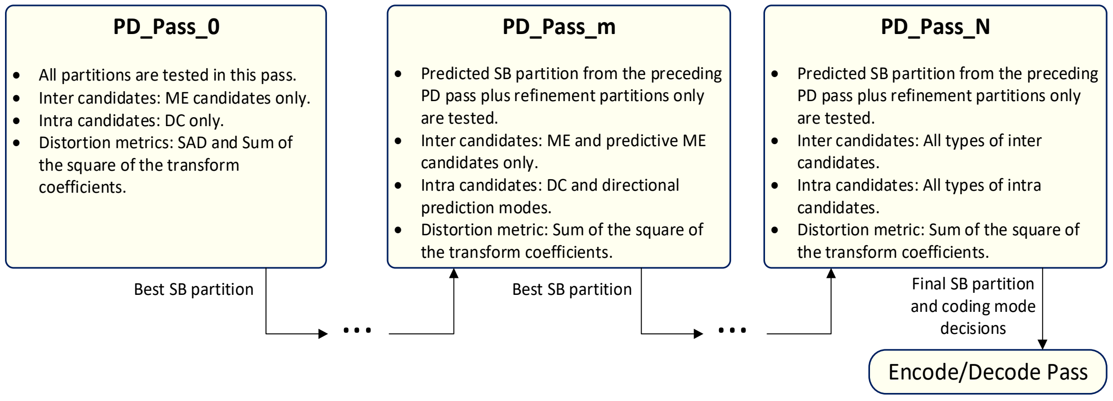

PD_Pass_0
All partitions are tested in this pass.
Inter candidates: ME candidates only.
Intra candidates: DC only.
Distortion metrics: SAD and Sum of the square of the transform coefficients.
PD_Pass_m
Predicted SB partition from the preceding PD pass plus refinement partitions only are tested.
Inter candidates: ME and predictive ME candidates only.
Intra candidates: DC and directional prediction modes.
Distortion metric: Sum of the square of the transform coefficients.
PD_Pass_N
Predicted SB partition from the preceding PD pass plus refinement partitions only are tested.
Inter candidates: All types of inter candidates.
Intra candidates: All types of intra candidates.
Distortion metric: Sum of the square of the transform coefficients.
Best SB partition
Best SB partition
Final SB partition and coding mode decisions
Encode/Decode Pass

Figure 7. Example of the processing details at each PD pass.

4. MULTI-STAGE/CLASS MODE DECISION

To further reduce the complexity of the SVT-AV1 encoder and add processing granularity to its multi-pass partitioning process outlined in the previous section, we employ multiple mode decision stages and multiple classes within each of the PD passes. The mode decision tasks performed in each PD pass are structured as shown in Figure 8. The input candidates to each stage are grouped according to classes. Example of classes could correspond for example to Intra, Palette, Inter (ME-based) and Inter (MVP-based) candidates. Multiple mode decision stages are then executed in each PD pass, where the complexity of the mode decision stages increases from MD_Stage_0 to MD_Stage_n_{MD}, due to the use of more accurate prediction tools and more accurate performance measures. Once the input candidates to a given mode decision stage are processed, only the best among the evaluated candidates are passed on to the next mode decision stage, hence reducing the number of candidates to be processed in the subsequent stage. As a result, some classes might also be retired (i.e., not considered in the subsequent mode decision stages) if the costs of all their corresponding candidates are relatively high. The candidate classes have been introduced to ensure that important types of prediction candidates are given a chance to survive to the final mode decision stage and to compete at that stage against the best from the candidates of the other classes. New candidate classes could also be added without affecting the survival chances of candidates from the other classes. Finally, it should be emphasized that the prediction and residual coding tools considered in mode decision are generally not-conformant tools, as the objective here is to produce partitioning and mode decisions, and not necessarily symbols and coefficients to be coded and transmitted in the bit stream, which is the task performed by the Enc/Dec process.

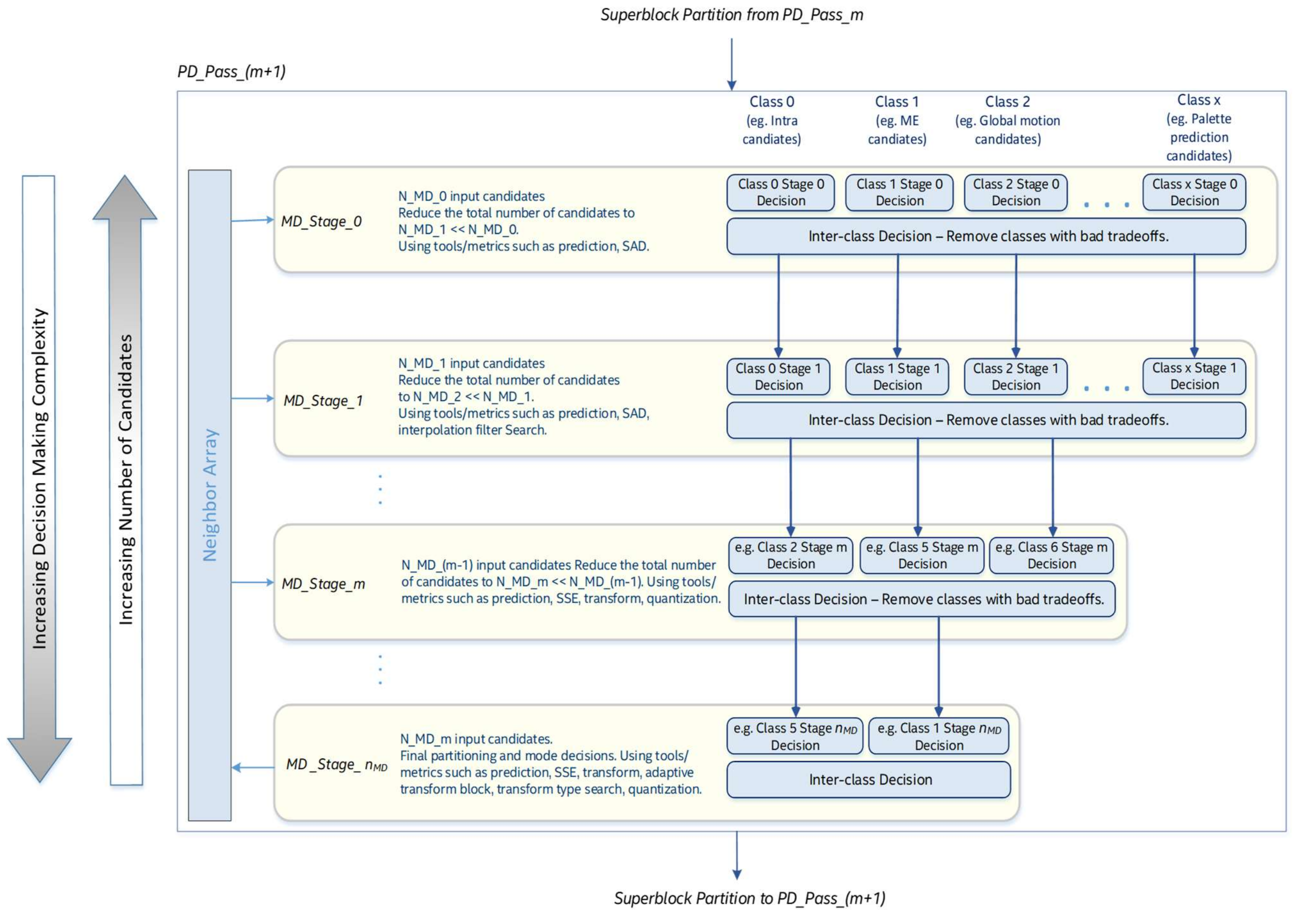

Superblock Partition from PD_Pass_m
PD_Pass_(m+1)
Class 0 (eg. Intra candiates)
Class 1 (eg. ME candiates)
Class 2 (eg. Global motion candidates)
Class x (eg. Palette prediction candidates)
Increasing Decision Making Complexity
Increasing Number of Candidates
Neighbor Array
MD_Stage_0
N_MD_0 input candidates
Reduce the total number of candidates to N_MD_1 << N_MD_0.
Using tools/metrics such as prediction, SAD.
Class 0 Stage 0 Decision
Class 1 Stage 0 Decision
Class 2 Stage 0 Decision
Class x Stage 0 Decision
Inter-class Decision – Remove classes with bad tradeoffs.
MD_Stage_1
N_MD_1 input candidates
Reduce the total number of candidates to N_MD_2 << N_MD_1.
Using tools/metrics such as prediction, SAD, interpolation filter Search.
Class 0 Stage 1 Decision
Class 1 Stage 1 Decision
Class 2 Stage 1 Decision
Class x Stage 1 Decision
Inter-class Decision – Remove classes with bad tradeoffs.
MD_Stage_m
N_MD_(m-1) input candidates Reduce the total number of candidates to N_MD_m << N_MD_(m-1). Using tools/ metrics such as prediction, SSE, transform, quantization.
e.g. Class 2 Stage m Decision
e.g. Class 5 Stage m Decision
e.g. Class 6 Stage m Decision
Inter-class Decision – Remove classes with bad tradeoffs.
MD _Stage_ nMD
N_MD_m input candidates.
Final partitioning and mode decisions. Using tools/ metrics such as prediction, SSE, transform, adaptive transform block, transform type search, quantization.
e.g. Class 5 Stage nMD Decision
e.g. Class 1 Stage nMD Decision
Inter-class Decision
Superblock Partition to PD_Pass_(m+1)

Figure 8. Mode decision flow in a single PD pass.

5. COMPLEXITY-REDUCTION OF PREDICTION/CODING ALGORITHMS

The AV1 specification introduced several features [6-16] that improve the coding efficiency of an AV1 encoder. Many of the normative SVT-AV1 encoder features, while being major contributors to the compression efficiency of the encoder, result in a substantially increased computational cost when fully enabled. Examples of such features include the support of as many as *eight* non-square partitions, and the support of as many as *seven* reference pictures. The following sections describe the SVT-AV1 algorithms developed to reduce dramatically the computational cost associated with such features, and to develop multiple speed-versus-quality tradeoffs for the encoder.

5.1 Efficient processing of partitions

For each square block size, SVT-AV1 supports as many as eight non-square partitions, in addition to the square block itself, as shown in Figure 5. A major challenge is to reduce the computational cost of processing the square block and its non-square partitions while minimizing the resulting degradation in quality. To address this challenge, the SVT-AV1 encoder employs a complexity-reduction algorithm that (1) generates the probabilities of the different non-square partitions and then uses these probabilities to allocate more computational resources towards the evaluation of the most-likely partitions, (2) employs the number of non-zero quantized residual coefficients of the square partition as a measure of prediction accuracy, and allocates less resources for the non-square partitions when the number of non-zero quantized coefficients is very low (such as zero), (3) analyzes the deviations between the rate-distortion costs of the three square/H/V partitions to help the encoder decide whether to test the three HA/HB/H4 and/or the three VA/VB/V4 partitions, and (4) reduces or removes redundancies between the non-square partitions.

a. Partition processing decision based on probabilities of partitions

The main idea here is to collect the statistics of the selected partitions while encoding the reference pictures. The statistics are then used to generate the probabilities of the possible partitions (including both square and non-square partitions). Such probabilities are used by the encoder to decide the partitions that will need to be tested, depending on the expected speed/efficiency of the encoder. In the following, the collection of the partition statistics is presented, followed by a discussion on the use of such probabilities to help the encoder decide which and how partitions should be tested.

i. Collection of statistics

Once the partitioning and coding mode decisions of an SB are finalized, the statistics of the decisions of interest (i.e., selected partitions) for the SB are then collected by accumulating the area of the blocks that belong to the selected partitions. The statistics for all SBs of the reference pictures are then accumulated to generate the picture-level average statistics. Two relevant variables for which statistics are collected in the SVT-AV1 encoder are explained next:

- Depth: The depth variable indicates whether the parent square block belongs to the predicted depth (i.e. size) as indicated by the best SB partition, or to depths that are either above or below the current depth.
- Parent square block partition: This variable takes any value from 0 to 8, and corresponds one of the block's partitions. As shown in Figure 5, a value of 0 corresponds to the parent square block as-is, 1 corresponds to the H partition, etc.

Before the start of the processing of SBs of a new picture, the probabilities of the partitions are computed based on the statistics of all previously processed reference pictures. Table 1 below shows an example of the probability distribution of the square and non-square partitions for the predicted depth (from the best SB partition of the previous PD pass) and the (-1/+1) refinement depths. Notice that the partitions of the predicted depth are more likely than those of the other two depths, and that the Square/H/V partitions are much-more likely than the other six partitions.

Table 1. Example of the probability distribution of the square and non-square partitions as a function of the depth.

Depth	Square	H	V	HA	HB	VA	VB	H4	V4
Predicted (depth -1)	0.08	0.06	0.04	0.03	0.01	0.01	0.01	0.01	0.00
Predicted depth	0.15	0.08	0.06	0.07	0.03	0.01	0.02	0.04	0.01
predicted depth +1	0.10	0.05	0.03	0.02	0.03	0.01	0.02	0.01	0.01

ii. Statistics-based processing of the partitions

Each of the nine partitions of a parent square block shown in Figure 5 has a probability of selection in the context of the parent square block's depth. For partitions that have high probabilities of selection, more accurate prediction and residual coding tools would be used in the evaluation of such partitions. On the other hand, partitions with low probabilities would be processed with low-accuracy but fast-prediction/coding tools. When very-high encoding speeds are desired, the encoder achieves the best tradeoffs when not testing the least-likely partitions.

b. Transform-coefficients-based allocation of computations

This component of the complexity-reduction algorithm consists of using the number of quantized residual coefficients (e.g., non-zero quantized coefficients) of the square partition as a measure of prediction accuracy. More specifically, when the number of non-zero quantized coefficients is very low, less computational resources are applied to the corresponding non-square partitions, through applying lower-accuracy-level prediction and residual coding tools to such partitions.

c. Cost-deviation-based non-square-partition reduction

This part of the complexity-reduction algorithm assumes that the Rate-Distortion (RD) cost of the parent square block and that of either the corresponding H or V partition (see Figure 5) are available. If the relative difference between the H partition cost and the parent square block cost is greater than a given threshold, the HA, HB and H4 partitions are not evaluated. Similarly, if the relative difference between the V partition cost and the parent square block cost is greater than a given threshold, the VA, VB and V4 partitions are not evaluated. The threshold used in this evaluation is made more aggressive (which would result in more non-square partitions being discarded) when higher encoding speeds are desired, and for the least likely partitions when the evaluation of the parent square block results in zero quantized coefficients. A further check is performed to determine whether to discard some additional non-square partitions. If partition H cost is much greater than partition V cost, then all three HA/HB/H4 partitions are not evaluated. A similar idea is applied to the VA/VB/V4 partitions.

d. Exploiting redundancies in the non-square partitions

As can be clearly seen in Figure 5, some blocks of different partitions have the same size, shape, position and neighbors. A list of such blocks is shown in Table 2. These blocks are referred to as redundant blocks, since only one of them would need to be evaluated. Once the prediction for a redundant block is generated, the resulting prediction information can be reused with any other block in the same group of redundant blocks.

Table 2. List of redundant and similar blocks of the partitions shown in Figure 5.

Redundant Blocks	**Similar Blocks**
1,8	2,7
3,14	4,13
5,11,25	6,15,26
	9,12,27
	10,16,28

Like redundant blocks, similar blocks share the same size, shape and position. The only difference is that similar blocks have different neighboring blocks. A list of such blocks is also shown in Table 2. Given that the neighbor information would be different from one block to another in the same group of similar blocks, reusing very specific prediction decisions for a given block for the rest of the blocks in the group might result in highly inaccurate results. However, it might be possible to make use of such information as guidance for the prediction for the rest of the similar blocks. For example, if the best prediction mode for a given block is intra DC prediction, it may not be a good idea to use the same prediction mode for the rest of the similar blocks, but it might be beneficial to indicate that inter prediction may not be needed in the rest of the similar blocks, and therefore inter prediction could be skipped/avoided for those similar blocks.

5.2 Reference pruning

For each block, reference picture pruning takes place in ME, based on the relative HME/ME distortion values, and in mode decision, based on the relative compensation prediction distortion values and/or temporal distance values, corresponding to the then-allowed reference pictures for such block. Further details are presented next.

a. Reference pruning in ME

The hierarchical motion estimation (HME) is performed using down-sampled versions of the source or reconstructed pictures to get a rough indication of the search center to use in full-pel ME. For each SB of the picture, all the reference pictures are considered to determine the best matching SB from each reference picture. The corresponding HME distortions are then sorted to determine the minimum HME prediction distortion. The relative difference between each of remaining distortions and the minimum distortion is computed, and the candidates whose relative distortion differences are greater than a given threshold are not considered further in the subsequent stages of the encoder pipeline. The surviving candidates are then passed to the full-pel ME stage to be used as search centers in full-pel search. The resulting full-pel predictions are subsequently processed in a similar way as was done in the HME stage to identify the best candidates to process in mode decision. Similarly, the reference pictures resulting in high distortion in ME are also not considered in the rest of the encoder pipeline. This reference pruning reduces the ME search computations, but the larger benefit is a more substantial reduction in mode decision computations.

b. Reference pruning in Mode Decision

For each block, and for each reference picture for the block, the distortions of the best ME candidates and all MV predictor candidates (e.g., NEAR, NEAREST) are computed in order to select the minimum-distortion candidate. The reference pictures are then ranked according to their minima distortions, and the reference pictures that yield the best prediction distortions are selected. The number of the best reference pictures used for the inter-prediction of the block would depend on the inter-prediction type/mode (e.g., compound, OBMC). Two other criteria used for pruning reference pictures are:

i. Inter-intra distortion: Another pruning step is based on comparing the inter-prediction distortion to the intra-prediction distortion. For every reference picture being considered for the current block, the corresponding inter-prediction distortion is compared to that of the best intra-prediction distortion for the block. If the inter-prediction distortion is larger, then the corresponding reference picture is not considered in subsequent inter prediction steps.
ii. Temporal distance: Another method for pruning of reference pictures for a block is to limit the number of allowed reference pictures based on temporal distance. For example, the closest reference pictures are far more likely to be selected by the encoder, and they should therefore be selected when other information is not available.

6. EXPERIMENTAL RESULTS

Initial software deployments of encoders based on newly ratified standards usually target Video-On-Demand (VOD) use cases, which generally employ the highest-video-quality software encoders in order to benefit as much as possible from the resulting bandwidth efficiency gains. In fact, VOD use cases currently represent the first major market for AV1-compliant software encoders. Therefore, the focus in this section is on the evaluation of the BD-rate-computations tradeoffs of the SVT-AV1 encoder, relative to those of the libaom encoder (another open-source AV1 software encoder) and a few popular non-AV1 open-source encoders, namely the x264 (AVC), x265 (HEVC) and libvpx (VP9) encoders. The software encoders are evaluated in the context of two VOD use cases, as follows:

- Premium VOD: The evaluation for this use case involves comparing the SVT-AV1 tradeoffs to those of the libaom reference AV1 (a.k.a. libaom) encoder [17] for 10-bit content at seven video resolutions.
- General VOD: These use cases include processing of User Generated Content (UGC) and other non-premium VOD. The evaluation for this case includes a comparison between SVT-AV1 and the above-mentioned open-source encoders (i.e., libaom, x264, x265 and libvpx) in terms of BD-rate-computations tradeoffs for 8-bit content at five video resolutions.

The rest of this section describes the performed tests with respect to each of the two use cases outlined above, followed by a presentation of the simulation results.

6.1 Premium VOD use case

In premium VOD applications, bandwidth efficiency gains are of most importance and the encoder's computational complexity is usually of secondary importance. Therefore, simulations that are representative of premium VOD use cases are performed following the methodology described in [18]. The resulting encodings will allow SVT-AV1 to be compared to libaom in terms of mainly BD-rate, but also in terms of computational complexity. The simulations involve encoding video shots at multiple resolutions and bit rates for various encoder presets. The experiments are performed following the steps outlined below.

a. Selection of the test video clips

A total of 14 clips from the Chimera and ElFluente episodes representing a variety of scenes and video complexities are selected to run these simulations. The clips can be found under the Netflix section in [19]. The clips, being originally in 4Kp60 resolution, are first spatially down-sampled to 1080p60-resolution clips, and only the first 60 frames of each clip, representing a one-second shot of each scene, are used.

b. Spatial down-sampling of the video clips

After the initial 4K-to-1080p down-sampling, the 1080p60 60-frame clips are down-sampled further to six other resolutions:1280x720, 960x540, 768x432, 608x342, 480x270 and 384x216. The down-sampling is performed using ffmpeg (version N-53260-ga37109d555-static) with the following command line, resulting in a total of 98 clips, or 14 clips for each of the seven resolutions:

- ffmpeg -y -i in.y4m -sws_flags lanczos+accurate_rnd+print_info -strict -1 -s:v <1280x720>   output.y4m

c. Encoder configurations

Both the SVT-AV1 and libaom encoders are configured to operate in the Constant Rate Factor (CRF) mode [20], the most-popular mode used to maximize coding efficiency. The following command lines are used:

- **Libaom (master branch, version 2.0.0-634-g62c8ebe48):** ./aomenc --passes=2 --verbose --end-usage=q --lag-in-frames=25 --auto-alt-ref=1 --kf-min-dist=128 --kf-max-dist=128 --cq-level=<20-63> --cpu-used=<0-6> -o out.bin   in.y4m
- **SVT-AV1 (ams-svt-01 branch, commit 16770fa):** ./SvtAv1EncApp --preset <0-6> -q <20-63> --keyint 128 --lp 1 -i in.y4m -b out.bin

In order to generate encodings at multiple bit rate values, nine CRF values [20,26,32,38,43,48,55,59,63] are used. The CRF values are specified in the command line using -q for SVT-AV1 and --cq-level for libaom. For SVT-AV1, ~2x-spaced presets (0-6) are used and are specified using the --preset option in the command line. For libaom, the presets (0-6) are used and specified using the --cpu-used option in the command line. Given the number of bit rates and resolutions considered in this experiment, a total of 882 encodings are generated for each preset for each of the encoders (14 clips * 7 resolutions * 9 CRF values). A text file listing all command lines is generated for every preset and every encoder. The command lines are sorted by input resolution (highest resolution first), then by CRF values (lowest CRF first). The same sorting is used for all encoders.

All encodings for this experiment are each performed on an AWS EC2 instance, specifically, a c5.18xlarge instance running Ubuntu Server 18.04 Linux OS over a dual Intel® Xeon® Platinum 8124M with 144GB of memory. Multi-threading and Turbo frequency are both enabled on these instances allowing the instance to access 72 logical cores with a maximum all-core turbo speed of 3.6 GHz [21]. Running the list of commands is done by invoking the ***parallel*** Linux tool and passing to it the command lines generated as explained above. The ***parallel*** tool would then schedule running the command lines by executing newer commands when older ones retire, while maintaining N command lines running at a time. N is chosen to be 72 in order to maximize the CPU utilization of the instance. Each set of encodings per preset per encoder is run independently while capturing the run time using the GNU ***time*** command as follows:

- /usr/bin/time --verbose parallel -j 72 < commands_encoder<x>_preset<n>.txt

d. Encoding results

Once all encodings are done, the resulting bitstreams are collected, decoded, and up-sampled to 1080p resolution using the ffmpeg command line shown below:

- ffmpeg -y -i in.y4m -sws_flags lanczos+accurate_rnd+print_info -strict -1 -s:v 1920x1080   output.y4m

Three performance metrics representing a measure of the Y component distortion between the up-sampled decoded clip and the initial clip at the 1080p resolution are generated using ffmpeg and libvmaf (v1.5.1). The performance metrics are Peak-Signal-to-Noise-Ratio (Y-PSNR), Structural Similarity Index (Y-SSIM), and Video Multimethod Assessment Fusion (VMAF). In order to pick the RD points that would result in the best-possible tradeoffs between quality, bit rate and resolution, all resulting bit rates and quality metrics across all resolutions for each clip are passed to a C sample application [22] that determines the convex hull points based on all available points. These points are chosen to allow the application to switch between encodings corresponding to different resolutions (based on the available bandwidth) while also maintaining the best possible video quality at a certain bit rate. With respect to the performed simulations, the input to this process is a set of 63 encoding results (7 resolutions * 9 CRF points) per video clip. The output of this process is a reduced set of points per video clip, representing the best RD tradeoffs. Figure 9 shows an example of this process corresponding to encodings for the Netflix_Aerial clip generated using libaom preset 0 (--cpu-used=0) and where the performance metric is Y-PSNR. The convex hull points are selected from encodings corresponding to different resolutions and are colored in burgundy in the graph.

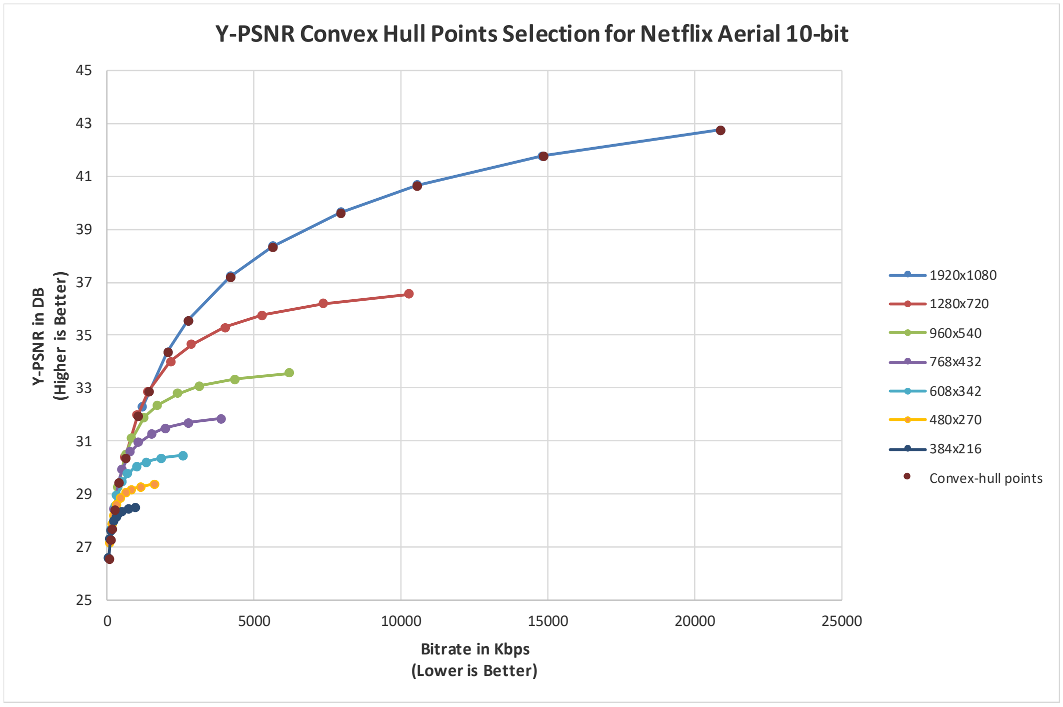


Figure 9. Y-PSNR RD Convex hull for Netflix Aerial 10bit with libaom preset 0 in 2pass mode.

The BD-rate results for all encodings are generated by comparing the resulting convex hull points for each of the video clips to those generated using libaom preset 0 (i.e. --cpu-used=0), which represents the reference encoder in this comparison. The BD-rate percentages are then averaged over all clips being tested, and an average BD-rate percentage is generated per encoder per preset, representing the percent BD-rate deviation of that preset as compared to libaom preset 0.

Figure 10 shows the results, with every point on the graph corresponding to an encoder preset. The y-axis represents the average BD-rate deviation of each encoder preset relative to that of libaom preset 0. The x-axis represents, on a logarithmic scale, the encoding time in seconds. The latter represents the sum of the "user time" and the "system time", where each of those two components of the encoding time is generated using the GNU utility ***time***. The encoding time represents the aggregate per preset of the encoding times of the 882 encodings. Note that the encoding time is not the same as the elapsed time (i.e. wall clock time), which records the time it takes to finish all of the encodings.

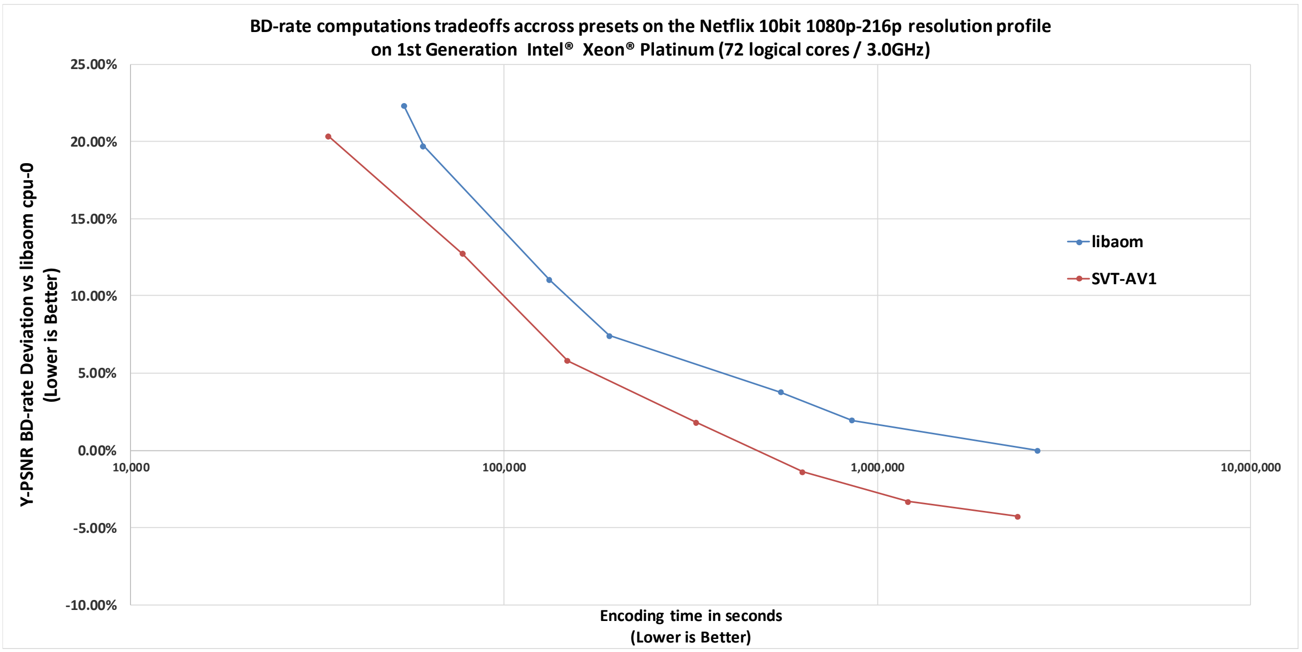


Figure 10. BD-rate (Y-PSNR) vs. encoding-time for SVT-AV1 and libaom in a premium VOD use case.

The results show that, in the highest video quality range, SVT-AV1 uses slightly less cycles than libaom while achieving a 4% BD-rate gain. Preset 2 of SVT-AV1 has a slight average BD-rate gain as compared libaom preset 0 while using approximately 24% of the cycles (i.e., a ~4x average encoding time advantage). These results demonstrate clearly the scalability feature of the SVT architecture, which scales (as intended) with both spatial and bit-depth resolutions, a feature that is highly desired in the premium VOD use cases. Another aspect to highlight in these results is that SVT-AV1 shows a consistent ~2x encoding time separation (or spacing) between the presets. Finally, Figure 11 shows similar results using Y-SSIM and VMAF, with an SVT-AV1 advantage that is slightly smaller for Y-SSIM and slightly larger for VMAF.

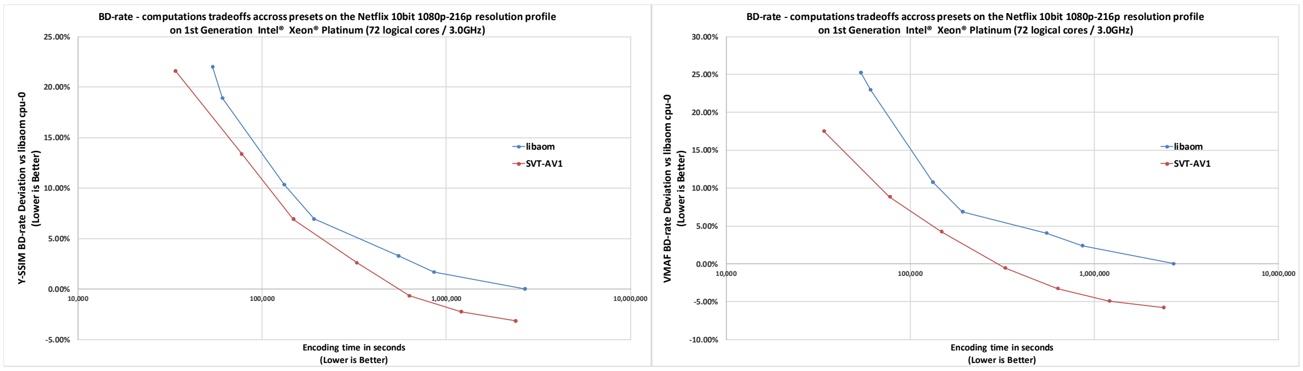


Figure 11. BD-rate (Y-SSIM/VMAF) vs. encoding time for SVT-AV1 and libaom in a premium VOD use case.

6.2 General VOD use cases

In the general VOD use cases, video compression efficiency is still an important objective. However, given the much larger number of encodings that usually need to be completed in such use cases, the computational complexity of the encoder becomes more important as compared to the premium VOD case.

Following the same methodology outlined in Section 6.1, more simulations are conducted to expand the comparisons to include more clips and more open-source software video encoders that are based on other video coding standards. The purpose of these simulations is to compare the BD-rate/computations tradeoffs of the SVT-AV1 encoder to those of available open source encoders that are currently being used in the target VOD applications. Unlike the premium VOD use case, the simulations for the general VOD use case make use of 8-bit content and fewer spatiotemporal resolutions. This section presents the changes in the testing conditions, as compared to those described in Section 6.1, and then discusses the results of the simulations.

a. Selection of the test video clips

The same 14 10-bit 60-frame Netflix clips used in the previous simulations are converted to 8-bit clips. The test content is expanded to add some video clips that have been open sourced by Facebook and Tencent for the future AV2 development work. More specifically, the added clips are AOV5, Skater227 (1080p), Wheat, WorldCup, WorldCupFarSky, and WorldCup_far [23]; they include a gaming clip and clips that could be categorized as UGC. A total of 20 8-bit clips (and more precisely, the first 60 frames of each clip) are therefore employed in the simulations.

b. Spatial down-sampling of the video clips

Using the same ffmpeg build and command line stated in Section 6.1, the twenty 1080p 8-bit clips are down-sampled to the following resolutions: 1280x720, 960x540, 640x360 and 480x270.

c. Encoder configurations

In addition to SVT-AV1 and libaom, the libvpx [24] (VP9), x264 [25] (AVC/H.264) and x265 [26] (HEVC/H.265) encoders are used in this evaluation. The CRF mode is also used in these simulations. The SVT-AV1 and libaom command lines are as shown in Section 6.1.c. The following are the command lines that are used with the added encoders:

- **x264 (v0.159.2991 1771b55):** ./x264 --preset <1-9> --threads 1 --tune psnr --crf <crf> --keyint 128 --min-keyint 128 --stats stat.stat -o out.bin in.y4m
- **x265 (v3.4+2-73ca1d7be377):** ./x265 --preset <0-9> --frame-threads 1 --no-wpp --tune psnr --crf <crf> --keyint 128 --min-keyint 128 --stats stat.stat -o out.bin in.y4m
- **libvpx (v1.8.2-196-ge53dc9f2e)**: ./vpxenc --passes=2 --verbose --end-usage=q --lag-in-frames=25 --auto-alt-ref=6 --kf-min-dist=128 --kf-max-dist=128 --cq-level=<crf> --cpu-used=<0-5> -o out.bin in.y4m

The command line for libvpx is chosen to be the same as the one used for libaom except for changing the "auto-alt-ref" option to be 6, following the recommendation in [5]. The x265 command line is also chosen based on the recommendations in [5]. In order to generate multiple bit rate points, nine CRF values are selected to perform the encodings. Due to the quantization range difference between AV1/VP9 and HEVC/AVC, the CRF values for the different encoders are as indicated below:

- AV1 / VP9 CRF values: [20,26,32,38,43,48,55,59,63].
- HEVC / AVC CRF values: [14,18,22,27,32,37,42,47,51]

The encodings for all allowed/supported presets for each open-source encoder are performed, with the exception of preset 0 (Ultrafast) for x264, where different encodings show very large BD-rate deviations. A total of 900 encodings are generated per preset for each of the encoders and for all supported bit rates and resolutions (20 clips * 5 resolutions * 9 CRF values). A text file listing all command lines is generated for every preset and for every encoder. The command lines are sorted by input resolution (highest resolution first), then by CRF values (lowest CRF first). The same sorting is used for all encoders.

All encodings are each performed on an AWS EC2 instance, specifically a c5.12xlarge instance running Ubuntu Server 18.04 Linux OS over a single 2nd Generation Intel® Xeon® Platinum 8275CL with 96GB of memory. Multi-threading and turbo frequency are both enabled on these instances allowing the instance to access 48 logical cores with a maximum all-core turbo speed of 3.6 GHz [21]. It is assumed that a smaller AWS instance is appropriate for the regular VOD use cases, since the latter usually require less compute cycles than the premium VOD use cases.

Following the same procedure used to generate the encodings as the one described in Section 6.1, the number of parallel jobs to be run on the instance is modified as shown in the command line below:

```
/usr/bin/time --verbose parallel -j 48 < commands_encoder<x>_preset<n>.txt
```

d. Simulation results

The BD-rates are generated in the same manner as in Section 6.1.d. The anchor used in the generation of the BD-rate data for SVT-AV1, libaom, x265, x264 and libvpx is libaom preset 0 (--cpu-used=0). Figure 12 shows the results of the simulations outlined above, where every point corresponds to an encoder preset. The y-axis represents the average BD-rate deviation of each encoding preset with reference to libaom preset 0. The BD-rate deviations are generated following the same method described in Section 6.1. The x-axis presents, on a logarithmic scale, the encoding time which is the sum of the user time and the system time, both of which are generated using the GNU utility ***time***. The encoding time represents the aggregate of the encoding time of each of the 900 encodings for the corresponding preset.

Figure 12 shows that although the AV1 encoders achieve significant bit rate savings, their encoding speeds are still not high enough to overlap with the x265/x264 fast speed ranges. The graphs do show though that for SVT-AV1, the fastest presets are already intersecting speed-wise with the Slower, Veryslow and Placebo presets of x265. Figure 13 shows a zoomed-in version of the graph to illustrate the area of the overlap. Figure 13 also reveals that, at a similar quality level to that of x265 Placebo and of x265 Veryslow, SVT-AV1 preset 6 takes only 10%/24% of the corresponding encoding times, respectively.

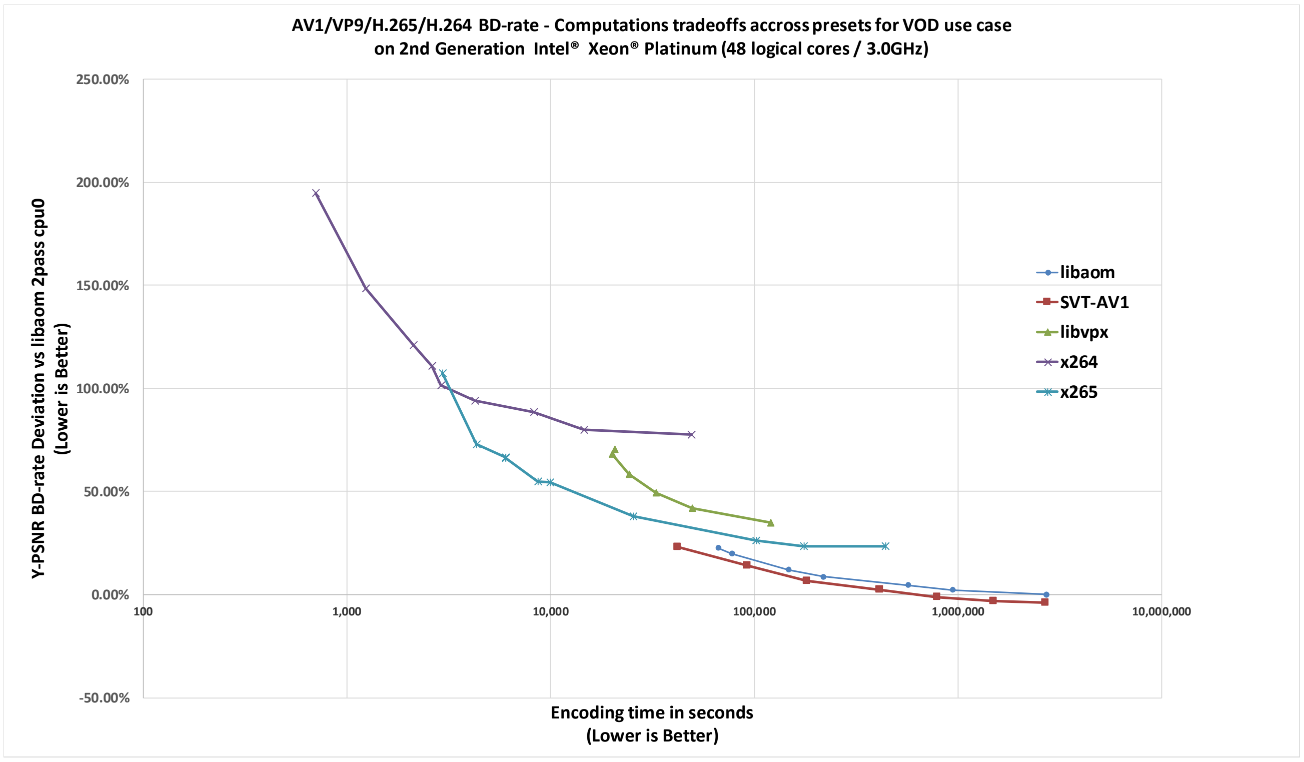


Figure 12. Y-PSNR BD-rate vs. encoding time for SVT-AV1, libaom, libvpx, x264, and x265 in general VOD.

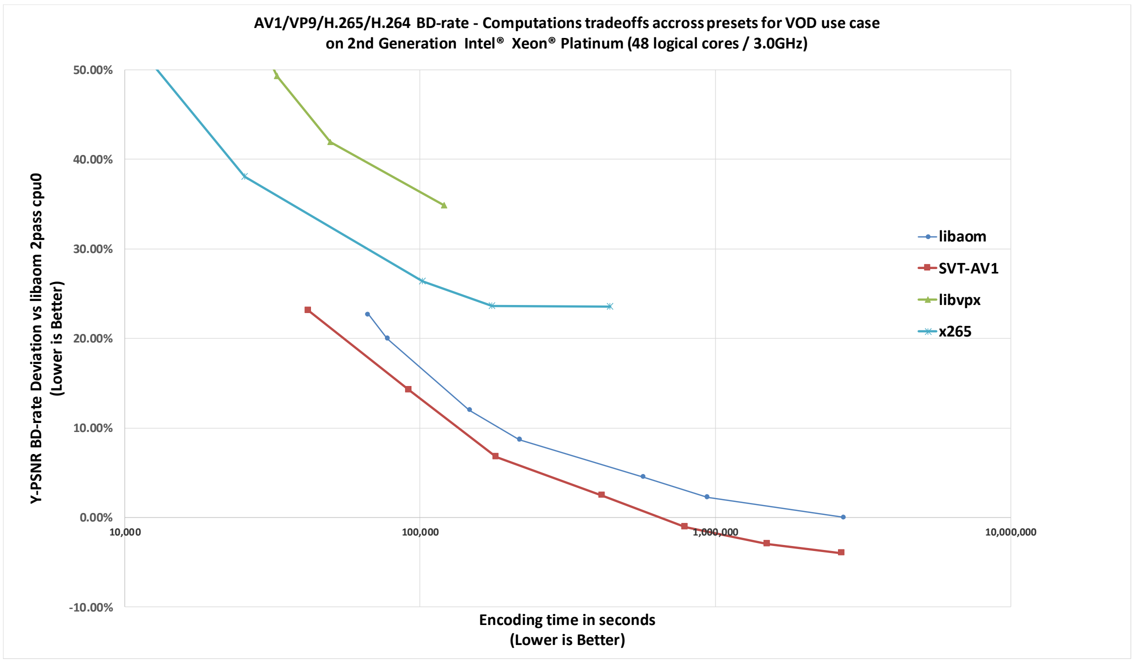


Figure 13. Zoomed-in version of Y-PSNR BD-rate vs. encoding time for SVT-AV1, libaom, libvpx, and x265 in general VOD.

If we were to generate a set of convex hull points across codecs for the chosen test set using Y-PSNR as a distortion metric, we would pick the SVT-AV1 presets for the slow-to-medium high-quality range, then the x265 Fast and Superfast presets for the medium speed range, then the x264 Veryfast and Superfast presets for the fast speed (low quality) range, as highlighted in Figure 14.

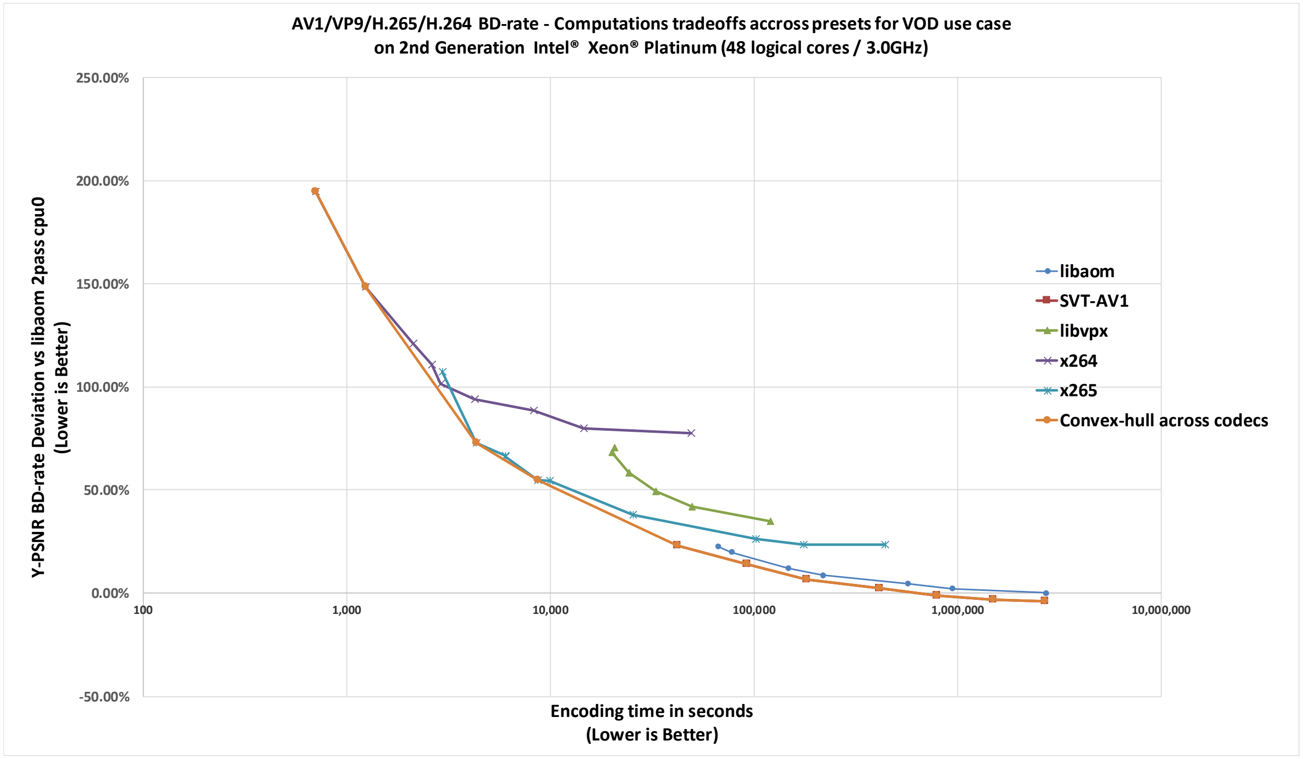


Figure 14 Convex hull points selection across codecs using Y-PSNR BD-rate vs. complexity tradeoff data.

Figure 15 shows the results of the simulations using Y-SSIM and VMAF; the results do not seem to change much the conclusions drawn using Y-PSNR as a metric, but show that both libaom and x265 can yield slightly better results using Y-SSIM, while SVT-AV1 can yield slightly better tradeoffs when using VMAF.

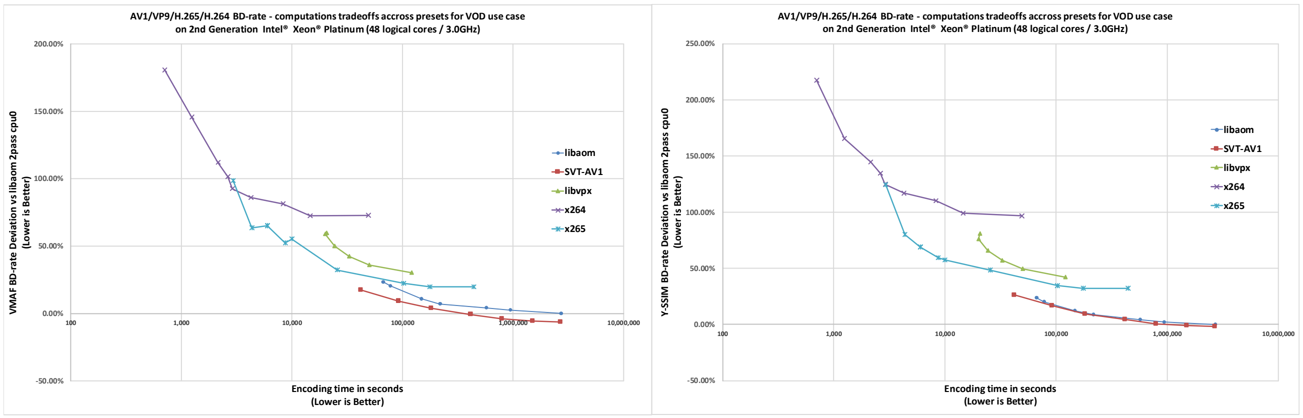


Figure 15. Y-SSIM/VMAF BD-rate vs. time for SVT-AV1, libaom, libvpx, x264, and x265 in general VOD.

To understand the results shown in Figures 12-15 for different quality ranges, and following the recommendations in [18], three VMAF-based quality intervals are defined, namely the Low Quality (LQ), High Quality (HQ) and Full Quality (FQ) intervals, corresponding to the VMAF score intervals [30,63], [63,96] and [30,96], respectively. The three-defined intervals are then mapped to the corresponding Y-PSNR and Y-SSIM ranges using a statistical analysis of such metrics across the video clips. For example, given a set of convex hull points for a clip, the bit rate yielding a VMAF value 30 is used to determine the corresponding Y-PSNR and Y-SSIM values resulting from using the same bit rate. All the resulting distortion values are then averaged over all of the clips to get the Y-PSNR/Y-SSIM values corresponding to a VMAF of 30. The same procedure is performed to obtain the average distortion values (needed to generate the three Y-PSNR/Y-SSIM intervals) that correspond to the VMAF scores 63 and 96. Following this method, the three values defining the LQ and HQ intervals are found to be (28.3, 32.5, and 40.1) and (7, 9.4, and 14.9), for Y-PSNR and Y-SSIM, respectively.

Figures 16-21 show how the results described in Figures 12-15 change when the BD-rate calculation of Y-PSNR, Y-SSIM, and VMAF are restricted to the LQ, HQ, and FQ quality ranges, respectively. We perform the BD-rate calculations through two phases, an interpolation between the RD points being compared, and an integration to measure the area between the two interpolated curves. The distortion range restriction is applied only at the integration step. This would help product teams that are interested in boosting the video compression efficiency performance in certain quality ranges identify which codecs might be best suited for their target application. The results shown in Figures 16-21 reveal that the tradeoff curves between libaom, x265 and SVT-AV1 get farther apart in the LQ range, while in the HQ range, the curves get closer especially when using Y-SSIM as a distortion metric. The recommendation of using SVT-AV1, then x265, then x264, as a result of the generated convex hull across these encoders remains valid across all quality ranges.

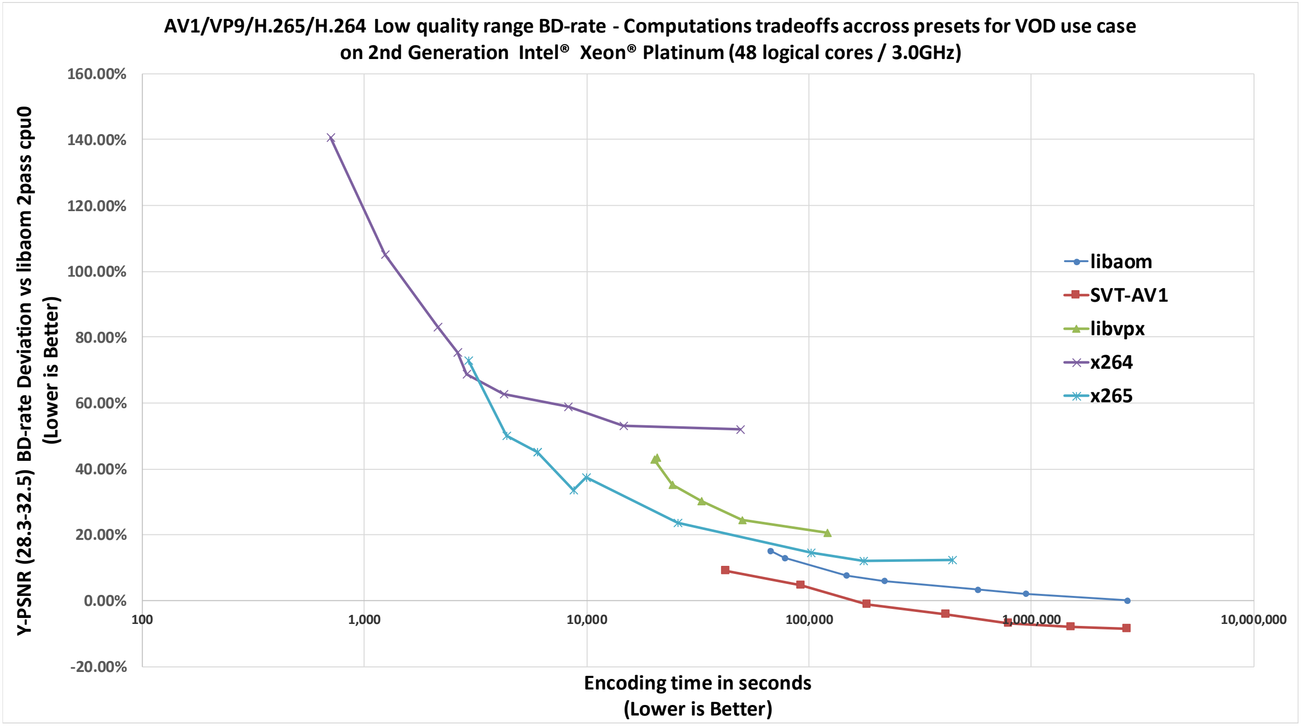


Figure 16. Y-PSNR BD-rate (Low-Quality range) vs. encoding time for SVT-AV1, libaom, libvpx, x264, and x265.

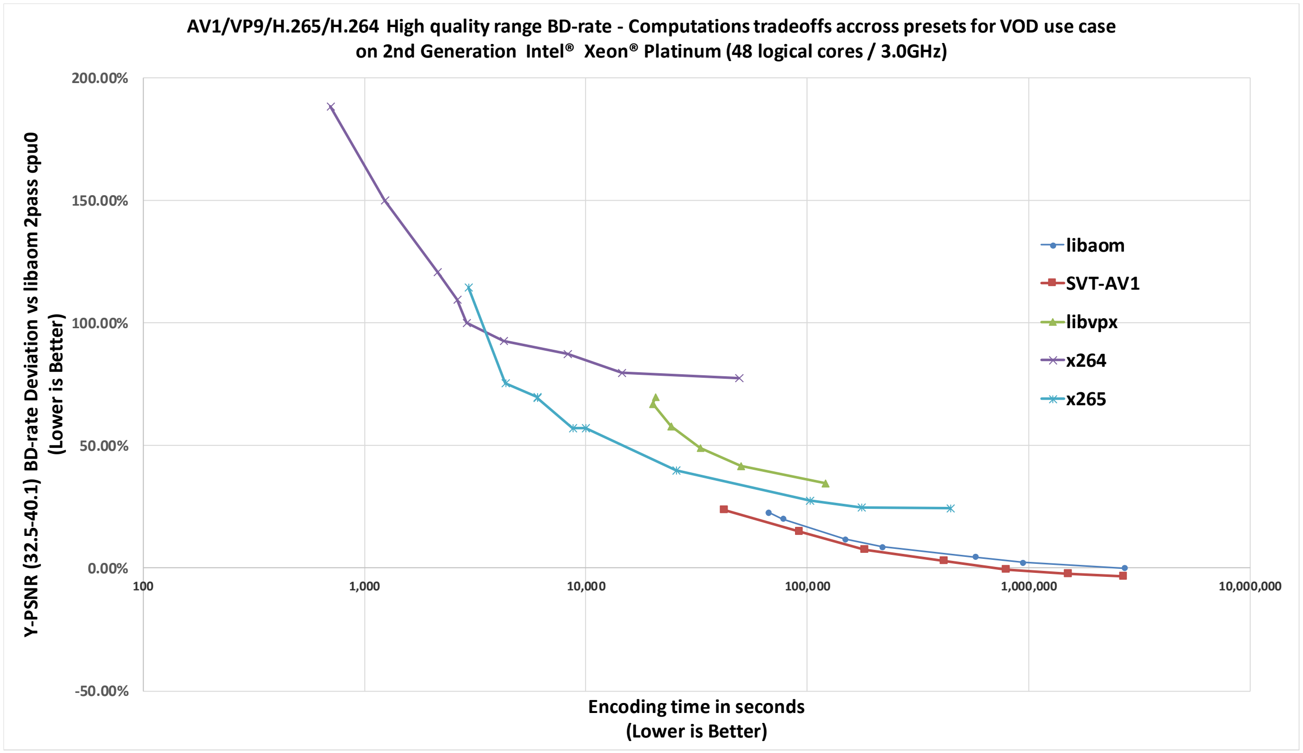


Figure 17. Y-PSNR BD-rate (High-Quality range) vs. encoding time for SVT-AV1, libaom, libvpx, x264, x265.

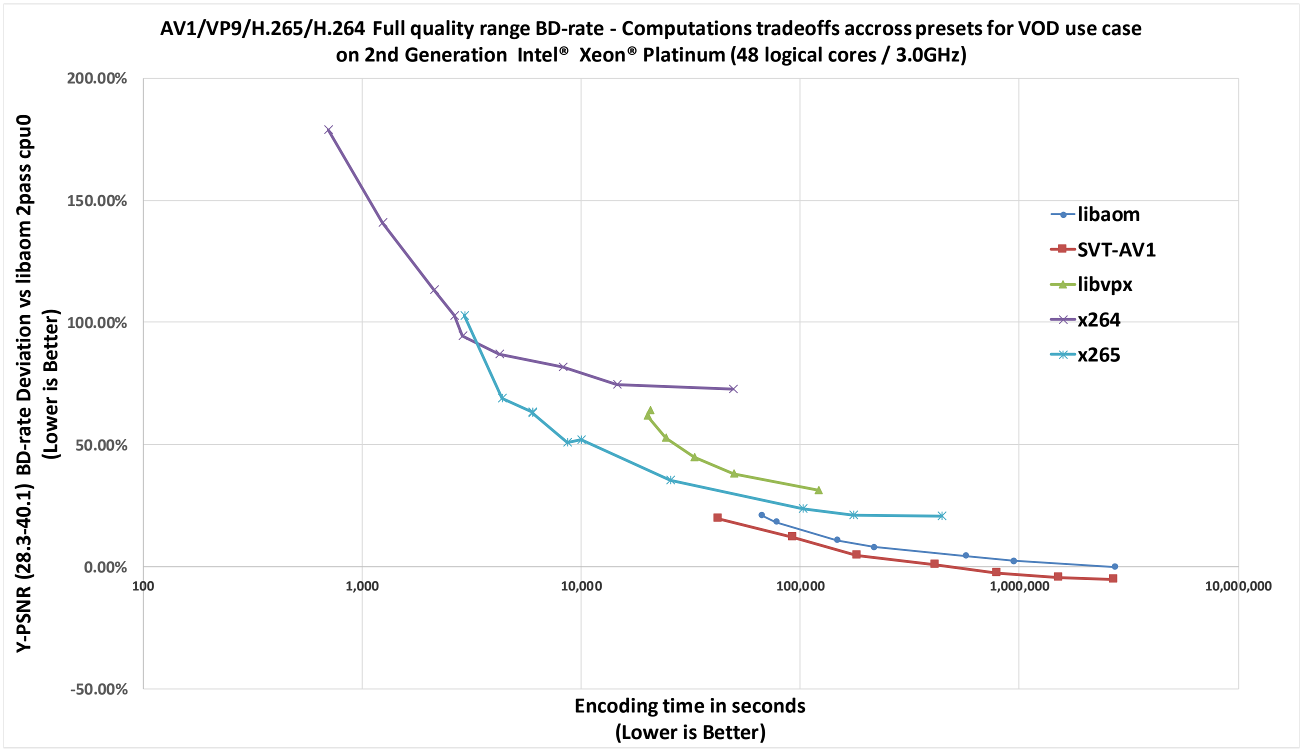
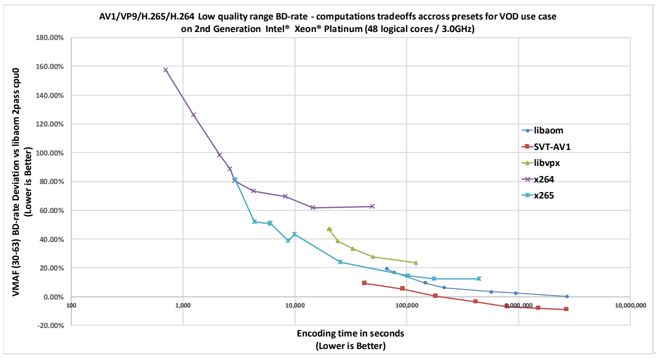


Figure 18. Y-PSNR BD-rate (Full-Quality range) vs. encoding time for SVT-AV1, libaom, libvpx, x264, and x265.



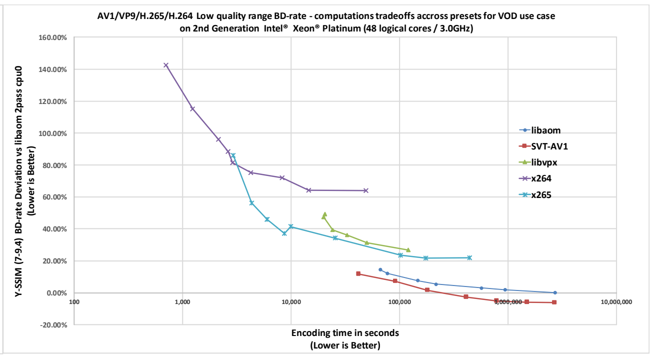
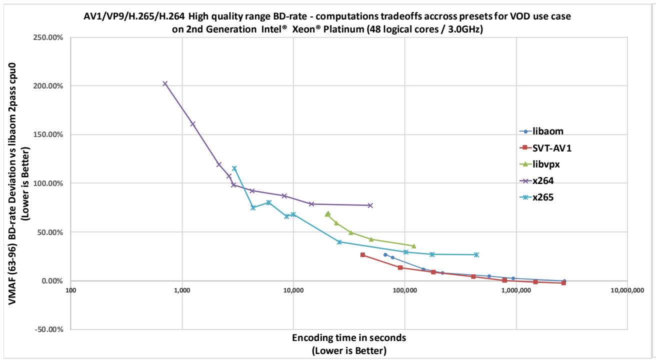


Figure 19. Y-SSIM/VMAF BD-rate (Low-Quality range) vs. time for SVT-AV1, libaom, libvpx, x264, and x265.



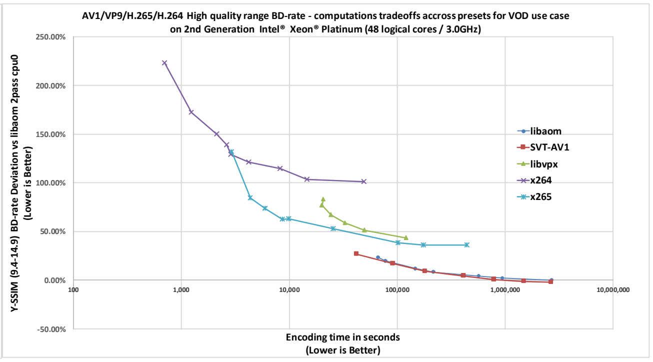
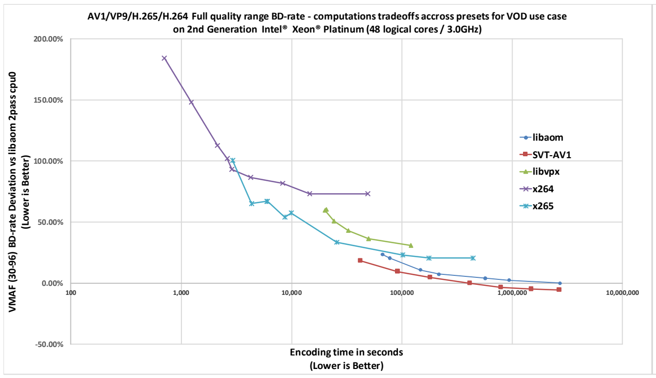


Figure 20. Y-SSIM/VMAF BD-rate (High-Quality range) vs. time for SVT-AV1, libaom, libvpx, x264, and x265.



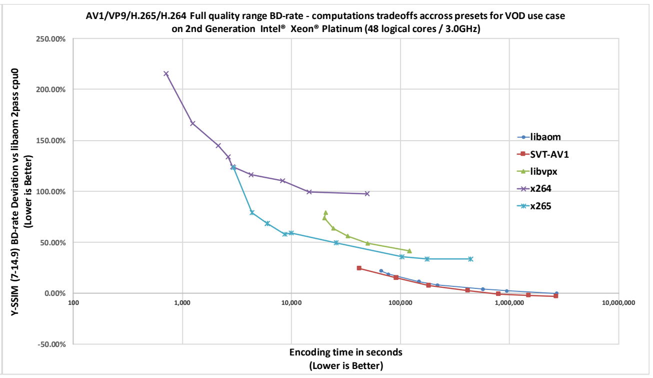


Figure 21. Y-SSIM/VMAF BD-rate (Full-Quality range) vs. time for SVT-AV1, libaom, libvpx, x264, and x265.

7. CONCLUSIONS

The SVT-AV1 encoder is an AV1-compliant software encoder that enables the efficient processing and encoding of multi-resolution video content. As its name implies, SVT-AV1 is scalable with respect to CPU cores, depth/spatiotemporal resolution, and computational resources, in the sense that it achieves consistently great BD-rate/computations tradeoffs for many multi-core CPUs and for multi-resolution video clips. The efficiency and scalability of the SVT-AV1 encoder are enabled through mainly architectural and algorithmic features. The SVT-AV1 encoder is evaluated in the context of premium and general VOD use cases. In premium VOD, SVT-AV1 showed significantly better BD-rate/computations tradeoffs than those of libaom. In general VOD use cases, the tradeoffs of SVT-AV1/libaom are quite similar. However, while the fastest presets of both encoders are still not fast enough to cover all the speed ranges of the x265, libvpx and x264 presets, it is clear that SVT-AV1 shows a greater promise of covering most of the x265/libvpx/x264 speed ranges. In fact, the evaluation results demonstrate that SVT-AV1 achieves consistently good tradeoffs for many VOD applications, ranging from premium VOD to UGC VOD. Preliminary experimental data with respect to faster presets show great promise that SVT-AV1 will eventually be the encoder of choice for a very-wide range of VOD applications.

Acknowledgments

The authors are very thankful to Ioannis Katsavounidis and David Ronca from the video infrastructure and video encoding teams at Facebook for providing guidance on the encoder evaluation methodology, and to Colton Cheung and Eric Lam for their help in running the experiments and scripting the workflow.

References

[1] Wiegand, T., Sullivan, G. J., Bjontegaard, G. and Luthra, A., "Overview of the H.264/AVC video coding standard", IEEE Transactions on Circuits and Systems for Video Technology, 13(7), 560-576 (2003).
[2] Sullivan, G. J., Ohm, J., Han, W. and Wiegand, T., "Overview of the high efficiency video coding (HEVC) standard", IEEE Transactions on Circuits and Systems for Video Technology, 22(12), 1649-1667 (2012).
[3] Mukherjee, D., Bankoski, J., Grange, A., Han, J., Koleszar, J., Wilkins, P., Xu, Y. and Bultje, R.S., “The latest open-source video codec VP9 - an overview and preliminary results,” Picture Coding Symposium, (2013).
[4] Alliance for Open Media, [online] Available: http://aomedia.org.
[5] Chen, Y., Mukherjee, D., Han, J., Grange, A., Xu, Y., Liu, Z., Parker, S., Chen, C., Su, H., Joshi, U., Chiang, C.-H., Wang, Y., Wilkins, P., Bankoski, J., Trudeau, L., Egge, N., Valin, J.-M., Davies, T., Midtskogen, S., Norkin, A., de Rivaz, P., "An Overview of Coding Tools in AV1: the First Video Codec from the Alliance for Open Media", SIP, 9, page 1-15 (2020).
[6] Trudeau, L. N., Egge, N . E. and Barr, D., "Predicting chroma from luma in AV1", *Data Compression Conference*, 374-382 (2018).
[7] Lin, W., Liu, Z., Mukherjee, D., Han, J., Wilkins, P., Xu, Y., Rose, K., "Efficient AV1 video coding using a multi-layer framework", *Data Compression Conference*, 265-373 (2018).
[8] Han, J., Xu, Y. and Bankoski, J., "A dynamic motion vector referencing scheme for video coding", *IEEE Int. Conference on Image Processing*, 2032-2036 (2016).
[9] Chen, Y. and Mukherjee, D., "Variable block-size overlapped block motion compensation in the next generation open-source video codec", *IEEE Int. Conference on Image Processing*, 938-942 (2017).
[10] Parker, S., Chen, Y. and Mukherjee, D., "Global and locally adaptive warped motion compensation in video compression", IEEE Int. Conference on Image Processing, 275-279 (2017).
[11] Joshi, U., Mukherjee, D., Han, J., Chen, Y., Parker, S., Su, H., Chiang, A., Xu, Y., Liu, Z., Wang, Y., Bankoski, J., Wang, C., Keyder, E., "Novel inter and intra prediction tools under consideration for the emerging AV1 video codec", Proc. SPIE 10396, (2017).
[12] Parker, S., Chen, Y., Han, J., Liu, Z., Mukherjee, D., Su, H., Wang, Y., Bankoski, J., Li, S., "On transform coding tools under development for VP10", Proc. SPIE, 9971, (2016).
[13] Han, J., Chiang, C.-H. and Xu, Y., "A level map approach to transform coefficient coding", *IEEE Int. Conference on Image Processing*, 3245-3249 (2017).
[14] Midtskogen, S. and Valin, J.-M., "The AV1 constrained directional enhancement filter (CDEF)", *IEEE Int. Conference on Acoustics Speech and Signal Processing*, 1193-1197 (2018).
[16] Mukherjee, D., Li, S., Chen, Y., Anis, A., Parker, S. and Bankoski, J., "A switchable loop-restoration with side-information framework for the emerging AV1 video codec", *IEEE Int. Conference on Image Processing*, 265-269 (2017).
[17] Alliance for Open Media AV1 Code Repository, https://aomedia.googlesource.com/aom/

[18] Katsavounidis, I., Liwei, G., "Video codec comparison using the dynamic optimizer framework," *Proc. SPIE 10752,* (2018)
[19] Xiph.org Video Test Media [Derf's collection], https://media.xiph.org/video/derf/
[20] Constant Rate Factor, https://trac.ffmpeg.org/wiki/Encode/H.264#crf
[21] Amazon EC2 C5 Instances, https://aws.amazon.com/ec2/instance-types/c5/
[22] Convex hull test app, https://github.com/ikatsavounidisFB/convex_hull
[23] AV2 Candidate Test Media, https://media.xiph.org/video/av2/
[24] webm libvpx project, https://github.com/webmproject/libvpx
[25] x264 project, https://www.videolan.org/developers/x264.html
[26] x265 project, https://bitbucket.org/multicoreware/x265/wiki/Home